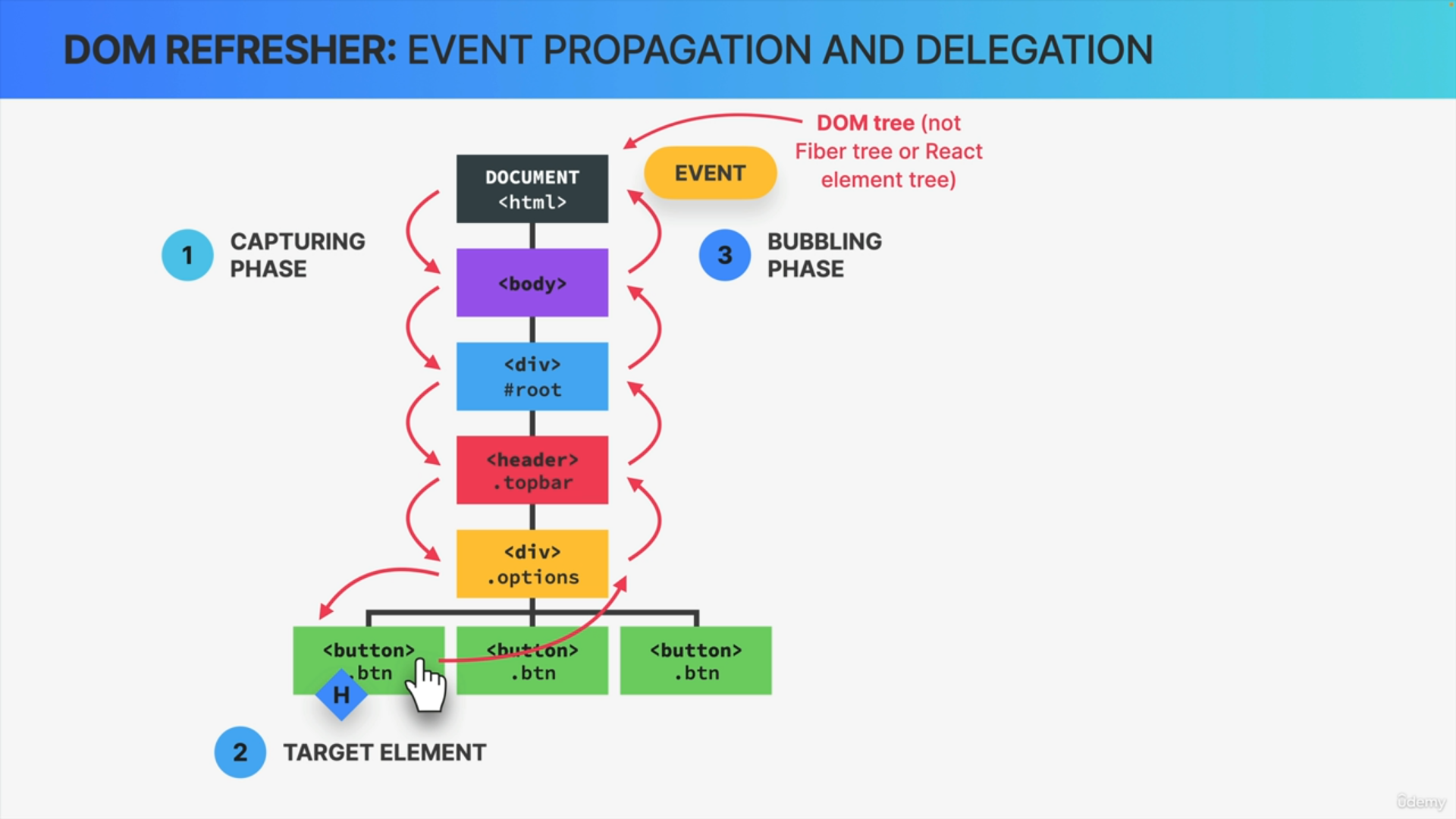
**Event Propagation**

When an event (e.g., a click) occurs, it follows these phases:

* Capturing Phase: The event travels down from the root of the document to the target element.
* Target Phase: The event reaches the target element where it can be handled.
* Bubbling Phase: The event bubbles back up from the target element to the root.

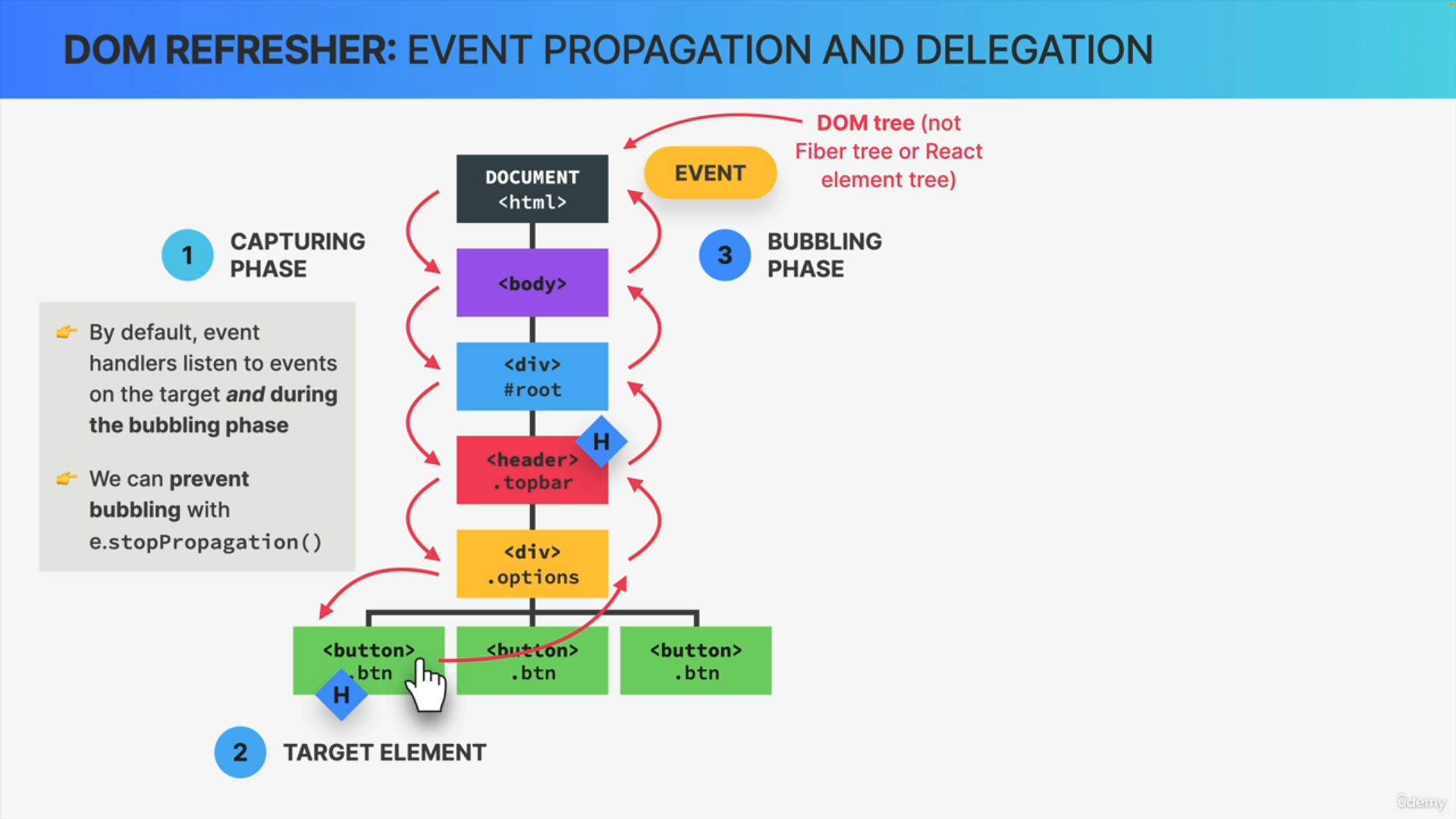


**Event Handlers**

By default, event handlers listen during the bubbling phase. Handlers on parent elements also execute if listening for the same event type.

**Stopping Propagation**

stopPropagation method can be used to prevent further bubbling.



function App() {

const handleButtonClick = (event) => {

console.log('Button clicked');

};

const handleDivClick = (event) => {

console.log('Div clicked');

};

return (

<div onClick={handleDivClick}>

<button onClick={handleButtonClick}>Click Me</button>

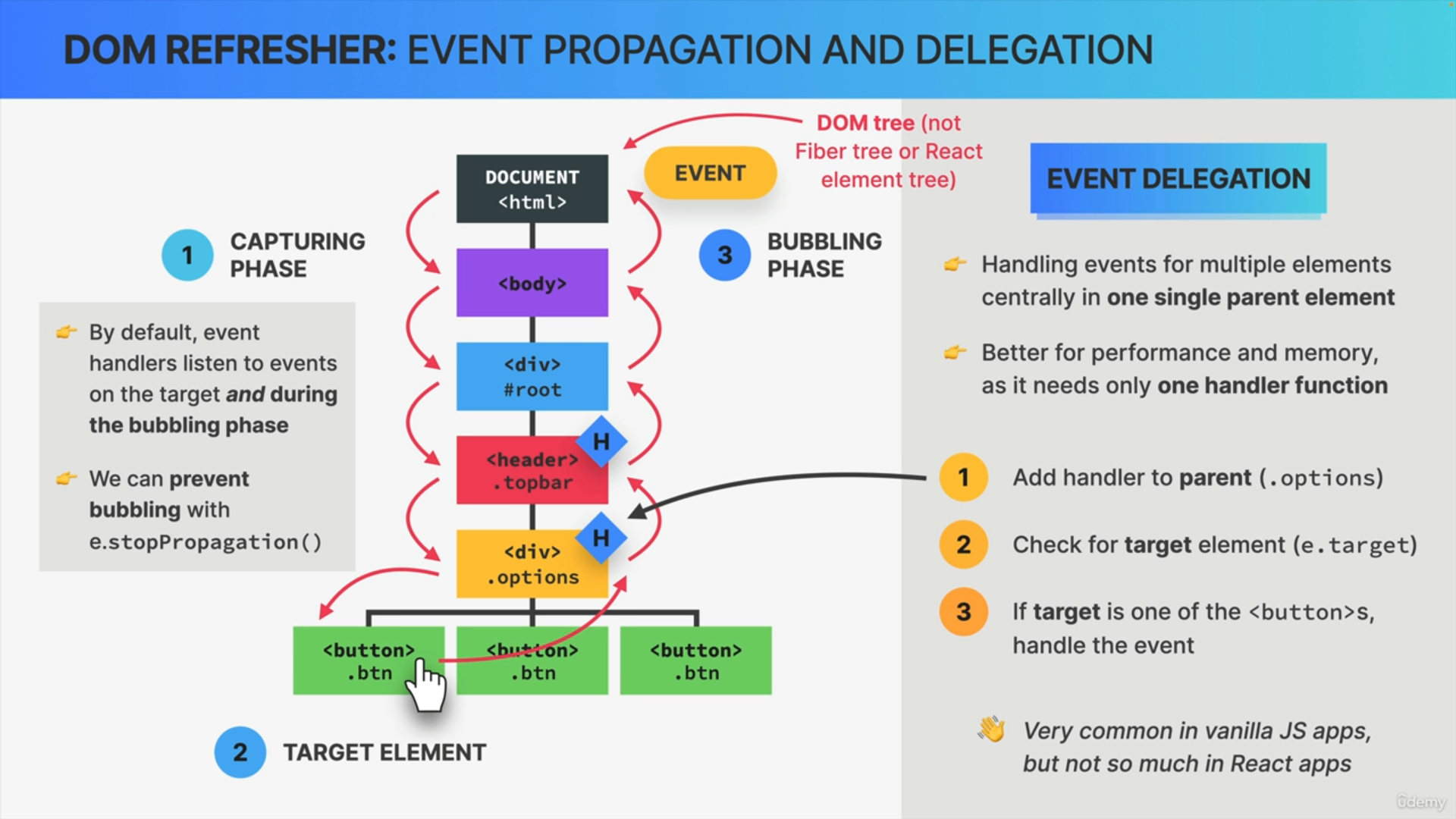
</div>

);

}

**Event Delegation**

This technique involves adding a single event handler to a parent element to manage events for multiple child elements, optimizing performance and memory usage.



function App() {

const handleClick = (event) => {

if (event.target.tagName === 'BUTTON') {

console.log(`Button ${event.target.textContent} clicked`);

}

};

return (

<div onClick={handleClick}>

<button>Button 1</button>

<button>Button 2</button>

<button>Button 3</button>

</div>

);

}

**Synthetic Events**

function App() {

const handleButtonClick = (event) => {

event.preventDefault(); // Prevents the default action

event.stopPropagation(); // Stops the event from propagating further

console.log('Button clicked');

};

const handleDivClick = (event) => {

console.log('Div clicked');

};

return (

<div onClick={handleDivClick}>

<button onClick={handleButtonClick}>Click Me</button>

</div>

);

}